SEUPD@CLEF: Team CLOSE on temporal persistence of IR systems' performance

Notebook for the LongEval Lab on Information Retrieval at CLEF 2023

Gianluca Antolini[1], Nicola Boscolo[1], Mirco Cazzaro[1], Marco Martinelli[1], Seyedreza Safavi[1], Farzad Shami[1] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract

This paper presents the work of the *CLOSE* group, a team of students from the University of Padua, Italy, for the *Conference and Labs of the Evaluation Forum (CLEF)* LongEval LAB 2023 Task 1 [1]. Our work involved developing an *Information Retrieval (IR)* system that can handle changes in data over time while maintaining high performance. We first introduce the problem as stated by CLEF and then describe our system, explaining the different methodologies we implemented. We provide the results of our experiments and analyze them based on the choices we made regarding various techniques. Finally, we propose potential avenues for future improvement of our system.

Keywords

Information Retrieval, Search Engines, Longitudinal Evaluation, Temporal persistence, Model Performance

1. Introduction

Recent research has shown that the performance of information retrieval systems can deteriorate over time as the data they are trained on becomes less relevant to current search queries. This problem is particularly acute when dealing with temporal information, as web documents and user search preferences evolve over time. In this paper, we propose a solution to this problem by developing an information retrieval system that can adapt to changes in the data over time, while maintaining high performance.

Our approach involves using the training data provided by the *Qwant* [2] search engine, which includes user searches and web documents in both French and English. We believe that this data will enable our system to better adapt to changes in user search behavior and the content of web documents.

The remainder of this paper is organized as follows: Section 3 describes our approach in more detail, including the different techniques we used. Section 4 explains our experimental setup, including the datasets and evaluation metrics we used. Section 5

presents our main findings and analyzes them based on the choices we made regarding various techniques. Section 6 goes into deep in analyzing results of our systems over test data and within time performances, making use of statistical tools such as ANOVA. Finally, Section 7 summarizes our conclusions and outlines potential avenues for future work.

2. Related Work

In the development of our IR system, we drew inspiration from various works and approaches.

This section provides an overview of the related works and methodologies that influenced our system's design and implementation.

2.1. *HelloTipster* Experiment

We initially looked to the *HelloTipster* [3] experiment, provided by Professor Nicola Ferro in his repository, as a starting point for implementing our IR system.

This experiment offered a basic *Lucene* [4] implementation that encompassed the main characteristics and functionalities of an IR system. By examining and understanding the *HelloTipster* experiment, we gained insights into the fundamental steps involved in an IR process.

2.2. Parser

A crucial step in the IR process is parsing the documents to extract relevant information.

In the legacy implementation, the content of each document was streamed into the analyzer, which tokenized the content and initiated the subsequent logical operations. While conventional IR practice often keeps this step simple to ensure generality and avoid overfitting, we incorporated additional actions to clean the document body of any scripting code. By removing such extraneous elements before passing the data to the analyzer, we ensured a cleaner input for token analysis.

Furthermore, since the documents were also available in *JSON* format, we modified the parsing phase to accommodate this format.

2.3. Analyzer

The analyzer component in the *HelloTipster* system was designed for English tokens.

However, we encountered the need to handle both English and French document collections. As a result, we introduced different configurations to the analyzer, allowing us to switch between them based on the specific collection being used.

With numerous configurable parameters within the analyzer, our workflow involved exploring the available filters in *Lucene* and conducting a trial-and-error process to identify the optimal configurations. Further details regarding the analyzer configurations will be discussed in the subsequent sections.

2.4. Searcher

The searcher component of our system required a parsing phase to extract the content of the topics. While the legacy code's parser worked with a slightly different format, we implemented a separate parsing section to align with the provided format. Building upon this, we incorporated two major improvements in the searcher component.

Firstly, we employed query expansion techniques to generate expanded queries for each topic, combining the original queries with their expansions by assigning greater weights to the original queries.

Secondly, we enhanced the search results by implementing a model that utilized sentence embeddings for results re-ranking.

By incorporating ideas and methodologies from these related works, we aimed to build a comprehensive and effective IR system that leveraged the strengths of existing approaches while addressing the specific requirements of our French and English collections.

3. Methodology

3.1. General Overview of Our IR System

In the development of our IR system, we followed the traditional *Y model*, represented in the figure below.
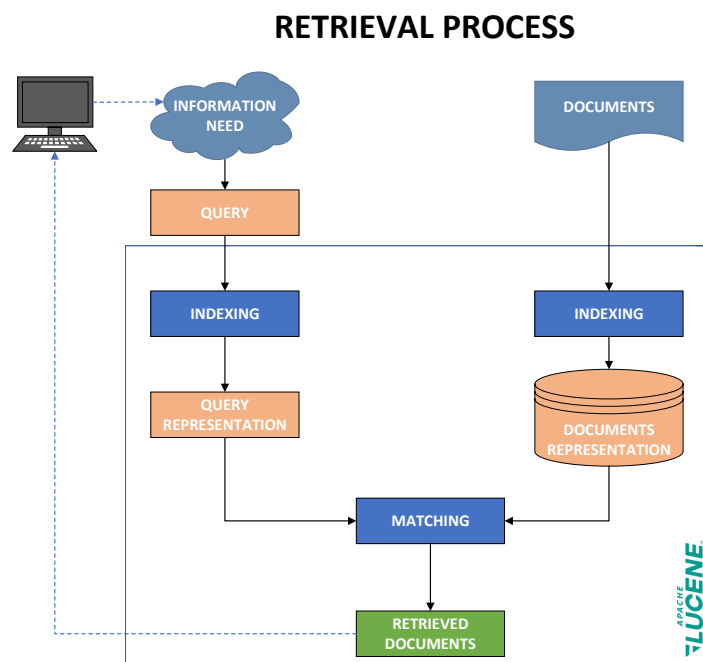
Figure 1: Apache Lucene Y Model.

The workflow of our system, starting from the train collection provided by *LongEval* [1], is as follows:

1. **Parsing**: The first phase consists of parsing the documents in the collection, which is a pre-processing operation performed to clean them from unnecessary noises. Since the collection is composed of web pages, the documents contain many leftovers like JavaScript scripts, HTML and CSS codes, HTTP and HTTPS URIs, and so on. The purpose of this phase is to ease the processing performed in the following phases.

2. **Indexing**: Each parsed document is then analyzed and indexed keeping only the necessary information. Indexed documents are composed of two fields: an *id* field, containing the identifier of the document in the collection, and a *content* field, containing the entire body of the document cleaned by the parsing and indexing phases.

3. **Query Formulation**: Topics are then parsed using the same analyzer used for documents, and used to formulate queries. For each topic, together with the already provided query, around 15 other query variants are generated (through the GPT model) by us and used altogether for searching relevant documents.

4. **Re-ranking**: Utilizing the sentence transformers model to determine the similarity between the document and the query. This similarity score is then multiplied by the BM25 score, resulting in new ranking scores for the documents. Then the retrieved documents are re-ranked based on these new ranking scores.
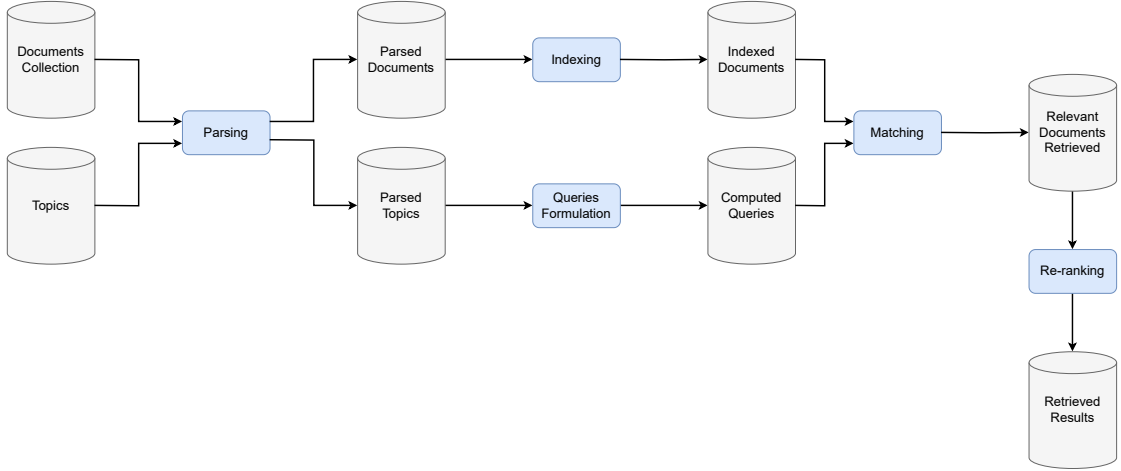
Figure 2: Workflow of the IR system implemented by *CLOSE*.
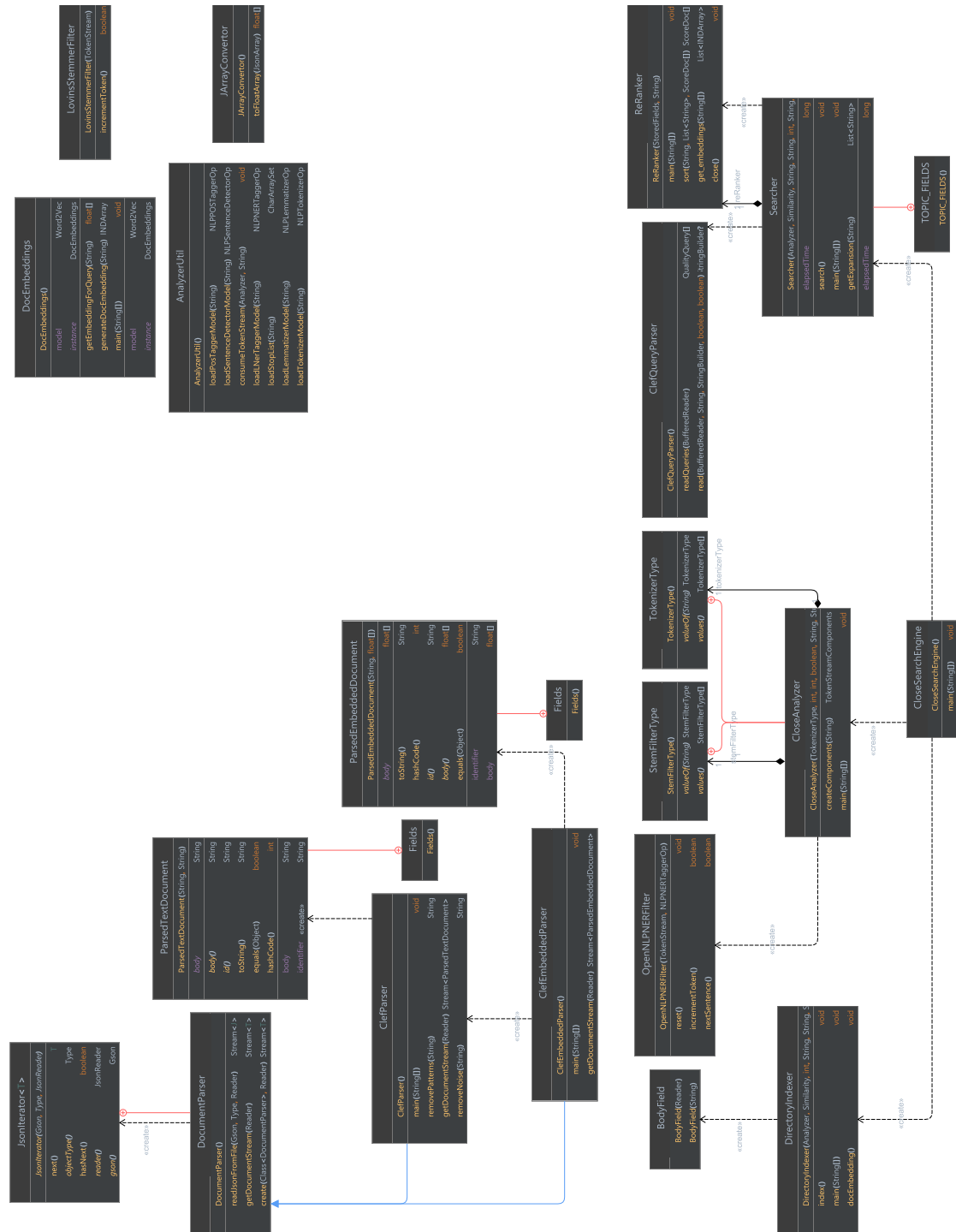
3.2. Class Diagram

Figure 3: Diagram of the classes implemented by *CLOSE* IR system

The class diagram of the system [3] retraces the Y model [1] of an IR system: in fact, it is possible to see it as an indexer class and a searcher class that is, in this order, called by the main class *CloseSearchEngine* of our system. The *Analyzer* class is instantiated before all, as it is used both from the two main branches stated before. Here many other Lucene (and Solr for NLP filters trials) tools are instantiated, as long as this component is responsible for analyzing tokens, and here most of our processing phase goes on: in particular, the tokenizer and the stemmer (and some NLP filters used in a trial).

The other connected component of the diagram is related to the parsing section: here, while walking on the file tree, the *Indexer* uses this component to generate, from a JSON document, an actual Java object (*ParsedTextDocument*) representing it with its fields. The *Searcher* class firstly parses the queries from the *Text REtrieval Conference (TREC)* format in a Lucene *QualityQuery* object, through our *ClefQueryParser* class. It also instantiates the *ReRanker*, responsible for the second-ranking phase.

Other utility classes are included but no longer used in our project.

3.3. Parser

As stated before, the documents in the collection provided by the CLEF *LongEval LAB 2023* [1] are essentially the corpus of web pages, to better represent the nature of a web test collection. From this, the need for performing a pre-processing phase of parsing the documents before analyzing and indexing them arises. In this phase, the documents are cleaned from all the residuals of codes not useful for our purposes. We first created an abstract class *DocumentParser* and then extended it by implementing a custom *ClefParser* class, which contains many functions for removing sundry types of noises that can be present in documents. This was the result of the trial and error approach we adopted for implementing this class:

- We started with our own read of a large statistical sample size of the documents in the collection to decide which types of noises needed to be removed.
- Then we implemented the parser and ran it.
- The results of the parsing were stored, and a sample of the parsed documents was analyzed to start this procedure again.
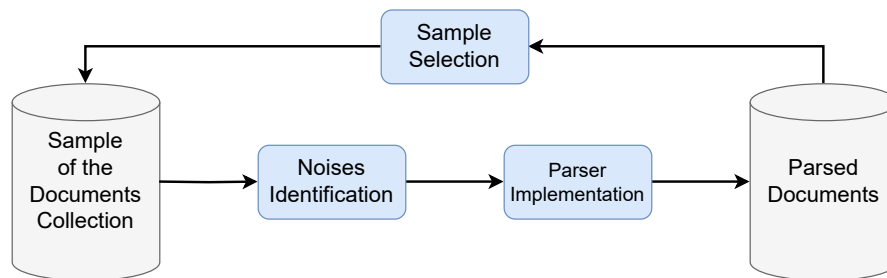
Figure 4: Workflow of the parser implementation.

The types of noises we tried to remove are the following:

- *JavaScript* scripts,
- *HTTP* and *HTTPS* URIs,
- *HTML* tags and *CSS* stylesheets,
- *XML* and *JSON* codes,
- Meta tags and document properties,
- Navigation menus,
- Advertisements,
- Footers,
- Social media handlers,
- Hashtags and mentions.

The final decision about the type of noises to effectively remove for our runs was the most crucial part of this process.

For establishing this we used a trial & error approach, and in the end, we decided to remove only the *JavaScript* scripts and the *HTTP* and *HTTPS* URIs. Regarding *URI*s, although these are usually important because they can contain valuable keywords, we noticed an improvement in *Mean Average Precision (MAP)* of almost 0.5 points by just removing them.

We also identified some patterns of words and symbols to remove:

- Two words separated by an underscore, like *word1_word2*
- Two words separated by a colon, like *word1:word2*
- Two words separated by a point, like "*word1.word2*"

We used *Regular Expressions* [5] to identify and remove these patterns.

The structure of the parsed document is defined in the *ParsedTextDocument* class, and it is composed of just two fields, as provided by CLEF:

1. *id*: the identifier of the document,
2. *body*: the (parsed) content of the document.

The constructor of *ParsedTextDocument* is the following:

```
/**
 * Creates a new parsed document.
 *
 * @param id the document identifier.
 * @param body the document body.
 */
public ParsedTextDocument(final String id, final String body)
```

Inside it, multiple controls about the validity and integrity of the parameters are performed, then an object of the class is instantiated.

3.4. Analyzer

The Analyzer is responsible for analyzing the extracted documents and preparing them for the Indexing and Searching phases. It does so by combining a series of techniques of text processing such as tokenization, stemming, stopword removal, and many more.

We extended Apache Lucene's Analyzer abstract class [6] by creating a custom class *CloseAnalyzer*, which is fully customizable by its parameters that can be chosen when creating an instance of the class. This has been done because we tried different settings and approaches to maximize the results and kept all the possible variations as optional settings. This *CloseAnalyzer* is passed as a parameter and then used by the *DirectoryIndexer* and by the *Searcher*.

The constructor of *CloseAnalyzer* accepts the following parameters:

- **tokenizerType**: used to choose between three standard *Lucene* tokenizers: *WhitespaceTokenizer* [7], *LetterTokenizer* [8], and *StandardTokenizer* [9].
- **stemFilterType**: the possible choices for the stemming types are four standard *Apache Solr* [10] filters: *EnglishMinimalStemFilter* [11], *KStemFilter* [12], *PorterStemFilter* [13], and *FrenchLightStemFilter* [14]. We also tried using *FrenchMinimalStemFilter* [15] and a custom filter called *LovinsStemmerFilter* based on a LovinsStemmer [16] implementation but decided to keep them commented as they didn't improve the results.
- **minLength** and **maxLength**: these are integers that simply specify the minimum and maximum length of a token, applying Lucene's *LengthFilter* [17].
- **isEnglishPossessiveFilter**: specifies whether to use Lucene's *EnglishPossessiveFilter* [18] or not. Of course, this can be useful when operating with the English dataset.
- **stopFilterListName**: with this parameter, it's possible to insert the path of an eventual word stoplist *.txt* file located in the *resources* folder. To do this we use Lucene's *StopFilter* [19] and a custom class called *AnalyzerUtil* that uses a *loadStopList* method to read and load all the stoplist words from the specified file. The stoplists we created are based on the standard ones but modified after inspecting the index with the *Luke* [20] tool. We have lists of different lengths and different ones for French and English.
- **Character nGramFilterSize**: if specified, this parameter is used to define the size of the n-grams to be applied by Lucene's *NGramTokenFilter* [21].
- **Word nGramFilterSize**: similar to the previous one, if used, this integer number indicates the shingle size to be applied by Lucene's *ShingleFilter* [22] that allows the creation of a combination of words.
- **useNLPFilter**: this boolean allows the use of Solr's [10] *OpenNLPPPOSFilter* [23] for Part-Of-Speech Tagging and of a custom class called *OpenNLPNERFilter* for Named Entity Recognition. To load the *.bin* models, which are located in the *resources* folder, we use two methods from *AnalyzerUtil*: *loadPosTaggerModel* and *loadNerTaggerModel*.
- **lemmatization**: specifies whether to use Solr's *OpenNLPLemmatizerFilter* [24] by loading a *.bin* model file in the *resources* folder using AnalyzerUtil's *loadLemmatizerModel* function.

- **frenchElisionFilter**: we applied this only when using the French dataset by adding Lucene's *ElisionFilter* [25] with an array of the following characters: 'l', 'd', 's', 't', 'n', 'm'.

On top of this, a *LowerCaseFilter* [26] is always applied.

We also tried Lucene's *ASCIIFoldingFilter* [27] and *SynonymGraphFilter* [28]. For the second one, only for the French Dataset, we used a *SynonymMap* [29] based on a *.txt* file containing French synonyms.

After different trials with different variations of the parameter, the following is the instance of the *CloseAnalyzer* we used:

```
final Analyzer closeAnalyzer = new
    CloseAnalyzer(CloseAnalyzer.TokenizerType.Standard, 2, 15, false,
    "new-long-stoplist-fr.txt", CloseAnalyzer.StemFilterType.French, null,
    null, false, false, true);
```

We have opted for the French dataset and by doing so we have the *StandardTokenizer*, 2 and 15 as minimum and maximum token length, we use *frenchElisionFilter*, *FrenchLight-StemFilter*, and a list of 662 French words as a stoplist. This stoplist has been built upon a popular French stoplist together with the most frequent stopwords in the collection. We didn't use any of the other parameters. We utilized the Gson library to efficiently parse JSON files that contained query expansions. By leveraging Gson's capabilities, we were able to seamlessly convert the JSON data into Java objects, enabling effortless manipulation and integration of the query expansions into our application.

3.5. Searcher

The purpose of the *Searcher* is to search through the indexed documents to retrieve relevant information based on user queries after analyzing them and to return a ranked list of documents that match the user's information needs.

Our implementation does so by accepting the following parameters:

- **analyzer**: in this case, an instance of *CloseAnalyzer*.
- **similarity**: we decided to opt for the *BM25Similarity* [30] function with the parameters *k1* and *b* tuned at 1.2 and 0.90.
- **Run options**: there are parameters for the index path, the topics path, the run path and the run name, the number of the expected topic (in our case 50), and the maximum number of documents retrieved (in our case 1000).
- **reRankModel**: this is the type of model used to do a Re-Ranking on the retrieved documents. In our case, we use a model called *all-MiniLM-L6-v2* [31], explained in the following subsection. If the parameter is set to null, no model is used and the documents are scored normally.

3.5.1. Query Expansion

When running the search function, one of the first actions performed is to generate new queries from the original ones by query expansion [32].

We created a Python script that, given the **.trec* topic file, generates all the expanded terms for each query and stores everything in a *.json* file called *result*, containing all the expansions.

We use *OpenAI's Text completion* [33] endpoints to generate the expansions, we can use our need as a prompt and the model will generate the result. We used the *davinci* model, which is the most powerful one, and we set the *temperature* parameter to 0.6, which is the value that gives the best results.

The sample result for prompt

"Expand the following query with num_expansions related terms or phrases for information retrieval (search-engine): query and the result should be in array format without any numbers at first [expanded_term1, expanded_term2, ...]"

is:

Query	text-davinci-002	text-davinci-003
antivirus comparison	1. Antivirus software 2. Antivirus protection 3. Best antivirus 4. Free antivirus 5. Antivirus for Windows 6. Antivirus for Mac	1. Antivirus Reviews 2. Antivirus Software 3. Antivirus Protection 4. Malware Protection 5. Virus Scanner 6. Online Security

The main difference between *davinci-text-002* and *davinci-text-003* is that the latter has been trained on a larger dataset, allowing it to generate more accurate results [34].

3.5.2. Query Boosting

Query boosting is a technique used to assign greater relevance to certain query terms or queries.

We tried the following approach that seemed to improve the overall results: when building the queries in the search function of the *Searcher* (3.5), for each query, a *BooleanQuery* [35] is built in the following way: after getting the query expansions, each of them is added to the *BooleanQuery* with the clause *SHOULD* (meaning that at least one of them must be satisfied) and a main query is added with the clause *MUST*, indicating that it must be satisfied.

This main query is boosted using Lucene's *BoostQuery* [36], with a boost value tuned at 14.68 multiplied by the number of expansions. We got this value by a trial & error approach we used to fine-tune this parameter.

3.5.3. Document Re-Ranking

Re-Ranking is the process of ranking documents retrieved by the search function of the *Searcher* (3.5). To accomplish this task, we utilized sentence transformers [37], a Python framework known for its state-of-the-art sentence, text, and image embeddings. Various models were experimented with, and the one yielding the best results was identified as *all-MiniLM-L6-v2* [31]. This particular model aims to train sentence embedding models using a self-supervised contrastive learning objective on vast sentence-level datasets, ultimately mapping sentences and paragraphs to a dense vector space of 384 dimensions.

To generate embeddings for both the documents and query retrieved by the search function, we loaded the *all-MiniLM-L6-v2* model and instantiated a *SentenceTransformer* object. To calculate similarity, we employed the widely used *cosine similarity* formula, which computes the similarity between two vectors. The formula is defined as follows:

$$\text{similarity} = \frac{t \cdot d_i}{|t||d_i|} \tag{1}$$

Here, t represents the query vector, and d_i denotes the vector of the i-th document.

Subsequently, we interpolated this similarity score with the BM25 score to improve the ranking of the documents. Among the different approaches we explored, the most successful one was as follows:

$$\text{rank} = \text{BM25\_score} \times \text{similarity} \tag{2}$$

To visualize the re-ranking operation's effectiveness, we plotted the results for 50 sample queries, as depicted in Figure 5.
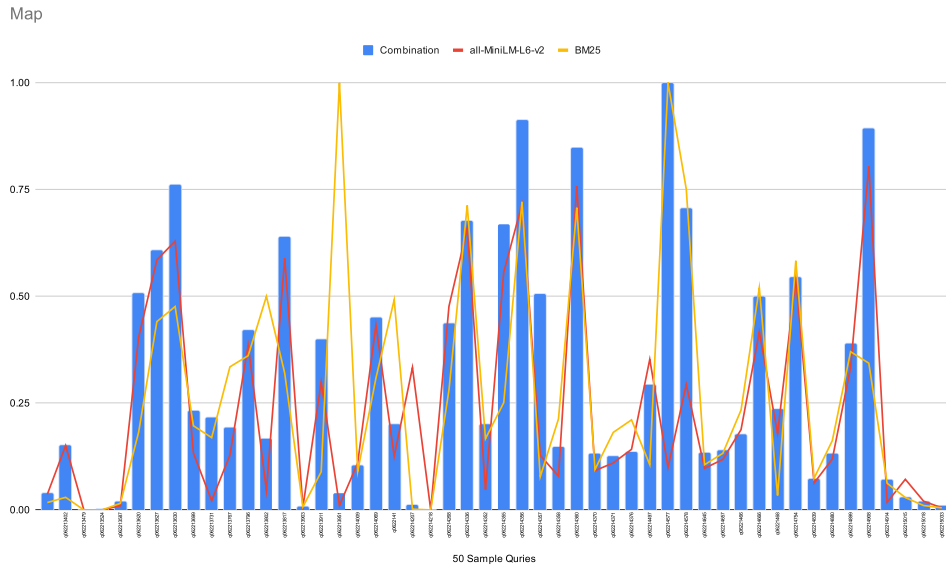
Figure 5: Plot illustrating the re-ranking operation performed on 50 sample queries

Finally, we sorted the documents based on the new rank and returned them.

4. Experimental Setup

4.1. Collections

We developed our model using a collection of 1,593,376 documents and 882 queries provided by *Qwant* search engine, available at https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-5010.
The collection contains information about user web searches and actual web pages corpora. The data was originally all in French but, for both queries and documents, an English translation is provided.

4.2. Evaluation Measures

To measure the effectiveness of our IR system we used the *trec_eval* executable by testing it with the resulting runs produced by the model and its different configurations.
We tracked improvements of the following evaluation measures generated by *trec_eval*:

- **num_ret**: number of documents retrieved for a given query.
- **num_rel**: number of relevant documents for a given query.
- **num _rel_ret**: number of relevant documents retrieved for a given query.
- **map**: Mean Average Precision, a measure of the average relevance of retrieved documents across all queries. Its range is [0-1], where 1 indicates that all relevant documents are ranked at the top of the result list.
  It's calculated as follows:
  $$MAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$
  where N is the total number of queries and AP_i is the average precision of *query$_i$* and calculated as:
  $$AP = \frac{1}{RD} \sum_{k=1}^{n} P(k)r(k)$$
  with RD the number of relevant documents for the query, n the number of total documents, $P(k)$ the precision at k and $r(k)$ the relevance of the kth retrieved document (0 if not relevant, 1 otherwise).
  A high MAP score indicates that the model effectively retrieves relevant documents for a wide range of queries.
- **rprec**: R-Precision is the precision score computed at the rank corresponding to the number of relevant documents for a given query.
- **p@5** and **p@10**: Precision at 5 and at 10 is the precision computed at the top 5 and 10 retrieved documents for a given query. They are calculated as follows:
  $$P(5) = \frac{1}{5} \sum_{k=1}^{5} r(k) \quad ; \quad P(10) = \frac{1}{10} \sum_{k=1}^{10} r(k)$$
  with $r(k)$ the relevance of the kth document.
  These two values can be useful to understand if the system retrieves relevant documents early in the result list, making it easier for the user to find the information needed.

- nDCG: it is a metric used to evaluate ranked lists. It measures the effectiveness of a ranking algorithm by considering item relevance. The nDCG is described by the formula:

$$\text{nDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}$$

where DCG (Discounted Cumulative Gain) calculates the cumulative gain of the top k items based on relevance scores, and IDCG (Ideal Discounted Cumulative Gain) represents the maximum achievable DCG. nDCG ranges from 0 to 1, indicating ranking quality.

4.3. Git Repository

The *git* repository of the project can be found at https://bitbucket.org/upd-dei-stud-prj/seupd2223-close/src/master/.
The code is available for reproducibility.

4.4. System Hardware

For the most time during the development of the system, every member of the group ran the model on its own system. It was after implementing the deep learning techniques that we decided to switch and start running the tasks using GPU to improve time performances.
The following are the specifics of the machines used after switching to computing also with GPU:

- CPU: Intel® Core™ i9-12900H 12th generation
- GPU: NVIDIA RTX™ A 2000 4GB GDDR6
- RAM: 16 GB SO-DIMM DDR5 4800MHz
- SSD: 512 GB M.2 2280 PCIe Gen4 TLC Opal

- CPU: Ryzen 7 1700 overclocked at 3.8GHz
- GPU: RTX 3070 TI 8GB GDDR6X
- RAM: 16GB DDR4 3000MHz
- SSD: Samsung evo 970 250GB

Parameter	Run 1	Run 2	Run 3	Run 4	Run 5
Token Filter	Porter-StemFilter	FrenchLight-StemFilter	FrenchLight-StemFilter	PorterStem-Filter	FrenchLight-StemFilter
Tokenizer	Standard	Standard	Standard	Standard	Standard
Length Filter	2-15	2-15	2-15	2-15	2-15
Stop Filter	"*long-stoplist.txt*"	"*long-stoplist-fr.txt*"	"*long-stoplist-fr.txt*"	"*long-stoplist.txt*"	"*new-long-stoplist-fr.txt*"
Lower Case Filter	Yes	Yes	Yes	Yes	Yes
Similarity	BM25	BM25	BM25	BM25	BM25
Query Expansion	No	Yes	Yes	Yes	Yes
Re-ranking	No	No	Yes	Yes	Yes

Table 1: Parameters used in the 5 different runs submitted to CLEF

5. Results and Discussion

In this Section, we provide some of the most relevant results we got during the development phase. We are considering five principal milestones that, within many different trials, led us to improve significantly our MAP score and the overall number of relevant documents actually retrieved.

First of all, given that we were provided with two different versions of the same document's *corpora*, our first idea was to try the English version.
We noticed that the best combination of basic IR tools was to use the *Porter* Stemmer [13], a length filter from 1 to 10, and a list of stop-word composed by some standard terms and more from the top 600 extracted from the index. The very first big milestone, that helped us to increment the MAP of around 3 points, from 10.1% to 13.1%, was the *JavaScript* code cleaner since we noticed by inspection that many documents were having these types of scripts inside.
Always by inspecting some documents and queries, and also considering that the original collection was the French version (translated then in English), we observed that the translation was very poor: by switching to French by just cleaning the *JS* code and some other minor cleaning tools, without even using an adequate stop-list and a correct stemmer for the French language, the MAP was increasing by +5%.
2 more MAP points were achieved with a stop list built for French in the same way we did previously for English, the *FrenchLightStemFilter* [14] as stemmer, and moving the length filter from 2 to 15 (as we noticed French tends to have longer words).
We tried some *Natural Language Processing (NLP)* techniques for English to see if there were improvements, and in this case, apply them to our main implementation for French with an appropriate model. The obtained results were not interesting, and also the computing time was definitely too costly. In particular, we tried to use Solr OpenNLP Part of Speech Filter [23] using the *en-pos-maxent Part of Speech (PoS)* tagger provided by *OpenNLP*. Another approach we tried and that carried an improvement was to use *Query expansion*: first we used some generative text models to expand our queries, then

metrics	run1	run2	run3	run4	run5
num__q	657	669	669	667	667
num_ret	646525	658446	658347	652222	657903
num_rel	2550	2611	2611	2603	2600
num_rel__ret	1772	2182	2191	1866	2232
map	0.1307	0.2022	0.2335	0.1856	0.2351
Rprec	0.1041	0.1697	0.1989	0.1654	0.2022
iprec_at__recall__0.00	0.2553	0.3499	0.4134	0.3584	0.4182
iprec_at__recall__0.20	0.2387	0.3324	0.3873	0.3353	0.3927
iprec_at__recall__0.40	0.1441	0.2316	0.2732	0.2066	0.2716
iprec_at__recall__0.60	0.0965	0.1786	0.1996	0.1388	0.2014
iprec_at__recall__0.80	0.0628	0.1178	0.1311	0.0887	0.1295
iprec_at__recall__1.00	0.0525	0.0954	0.1031	0.0704	0.1028
P_10	0.0848	0.1296	0.1435	0.1126	0.1432
P_100	0.0186	0.0256	0.0268	0.0222	0.0268
P_1000	0.0027	0.0033	0.0033	0.0028	0.0033
recall_10	0.2166	0.3352	0.367	0.2849	0.3621
recall_100	0.4718	0.6426	0.6714	0.5536	0.6723
recall_1000	0.6816	0.8192	0.8218	0.7004	0.8392
ndcg	0.2719	0.3655	0.3924	0.3291	0.3982
ndcg_cut_5	0.1285	0.1908	0.2232	0.1854	0.2269
ndcg_cut_10	0.1609	0.2426	0.2739	0.2227	0.2758
ndcg_cut_100	0.2351	0.3349	0.3652	0.3016	0.3678
ndcg_cut_1000	0.2719	0.3655	0.3924	0.3291	0.3982

Table 2: results for systems (top-1000 documents), on the Train collection and the train query set of LongEval.

we decided to weight different query scores by boosting the original one linearly with respect to the number of expansion used. This made us gain an extra MAP point.
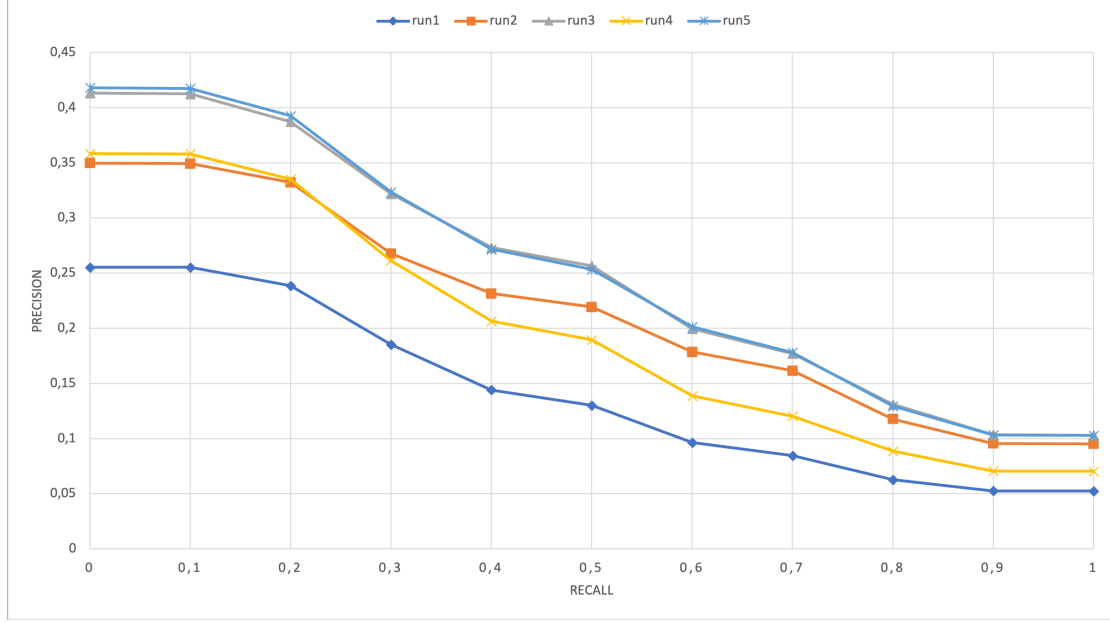
Figure 6: Recall and precision graph

We try to generate the embeddings for each document based on word2vector, we use a pre-trained word2vec model *frWac_no_postag_no_phrase_500_cbow_cut100* [38] for French. Then we calculate the embedding for each document and index them as KnnFloatVectorField in Lucene and use KnnFloatVectorQuery [39] for searching the query to find the *k* nearest documents to the target vector according to the vectors in the given field, but the results (overall MAP 0.08) were not satisfying, being worse than the case of indexing and searching without embeddings.

We then tried to combine different similarities rather than using the classic *BM25Similarity*: we tried to use the Lucene *MultiSimilarity* [40], that allows combining the score of two or more similarity scores, but it does not allow to tune the weights. Then, we tried to reimplement the *MultiSimilarity* class with tuning options, but the results were always lower than the standard *BM25Similarity*. Some minor improvements came up by fine-tuning the document-length normalization *b* parameter and the term frequency component *k1* parameter of the *BM25Similarity*.

The last main implementation we did, was to use some *Re-ranking* techniques to improve the results of the first retrieval phase. We tried to use the *SBERT* model [41], which is a pre-trained model for sentence embeddings, and we used it to calculate the similarity between the query and the document. We tried to use different distance metrics such as *CosineSimilarity* [42] and *ManhattanDistance* [43] for calculating the similarity, but at the end of the day, *CosineSimilarity* is much better than others. Finally, we sort the documents based on merging the BM25 score and similarity score into one score by multiplying them together.

Lastly, some minor adding were set on the *Analyzer* (see Section 3.4) by implementing the Lucene *ElisionFilter* (for French) [25], which aims to remove apostrophes articles and prepositions from tokens (for example, *m'appelle* and *t'appelle* become the same token *appelle*).

5.1. Test set results

heldout						
run	language	type	map	p@10	NDCG	recall
run2	FR	QUEREXPANSION	0.2029	0.1367	0.3725	0.8312
run3	FR	RERANKING	0.2595	0.1541	0.4166	0.8348
run5	FR	SBERT_BM25	0.2675	0.1561	0.4318	0.8726
run1	EN	JSCLEANER_BM25	0.1299	0.0897	0.2674	0.6381
run4	EN	RERANKING_ENGLISH	0.1822	0.1122	0.3113	0.6279

Table 3: results for systems (top-1000 documents), on the Train collection and the heldout query set of LongEval.

Short term						
run	language	type	map	p@10	NDCG	recall
run2	FR	QUEREXPANSION	0.2215	0.1326	0.3800	0.8164
run3	FR	RERANKING	0.2511	0.2171	0.4073	0.8142
run5	FR	SBERT_BM25	0.2540	0.1497	0.4142	0.8360
run1	EN	JSCLEANER_BM25	0.1438	0.0902	0.2746	0.6566
run4	EN	RERANKING_ENGLISH	0.1956	0.1145	0.3311	0.6804
Long term						
run	language	type	map	p@10	NDCG	recall
run2	FR	QUEREXPANSION	0.2067	0.1423	0.3745	0.8312
run3	FR	RERANKING	0.2388	0.1555	0.4071	0.8336
run5	FR	SBERT_BM25	0.2437	0.1594	0.4148	0.8540
run1	EN	JSCLEANER_BM25	0.1450	0.0975	0.2866	0.6906
run4	EN	RERANKING_ENGLISH	0.1930	0.1258	0.3391	0.7119

Table 4: results for systems (top-1000 documents), on the Test collection and the test query set of LongEval.

6. Statistical Analysis

In this section, we delve into a comprehensive analysis of the retrieval effectiveness for our 5 different runs submitted to CLEF on both the French and English collections (see Table 1). The analysis aims to evaluate the performance of different runs and provides insights into the effectiveness of the system in retrieving and ranking relevant documents.

We begin by analyzing the results obtained from the French collection. The overall *Normalized Discounted Cumulated Gain (nDCG)* and MAP comparison allows us to gain an initial understanding of the performance differences between runs 2, 3, and 5, considering short-term, heldout, and long-term evaluations. By examining the nDCG scores and *Relative nDCG Drop (RnD)* values, we can identify the run that demonstrates the highest effectiveness in retrieving relevant documents.

Furthermore, we employ two-way *ANalysis Of VAriance (ANOVA)* tests to investigate the significance of the observed differences in the long-term and short-term evaluations. This analysis provides valuable insights into the performance of the IR system on the French collection, guiding us in identifying areas for improvement.

Subsequently, we shift our focus to the analysis of results obtained from the English collection. We conduct an analogous evaluation, comparing the performance of runs 1 and 4 in the same way as stated above for the French collection.

The analyses and discussions provided in the next subsections aim to give a better understanding of the performance of the developed IR system and serve as a basis for optimizing its retrieval performances.

6.1. Analysis of Results on the French Collection

6.1.1. Overall *nDCG* Comparison

The overall nDCG comparison of Runs 2, 3, and 5 on the French collection is presented in Figure 7. This boxplot provides valuable insights into the performance of each run, with the bars representing the nDCG scores for each run. The color coding distinguishes the different evaluation periods, with short-term depicted in green, heldout in orange, and long-term in pink. The delta values within the boxes represent the RnD of each run compared to the heldout run.
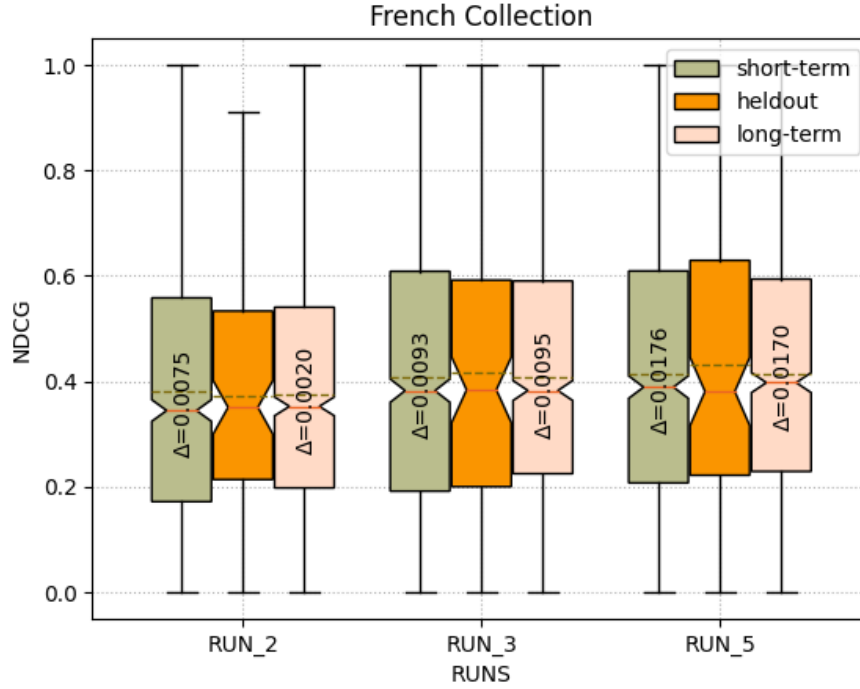
Figure 7: Overall nDCG Comparison for Runs 2, 3, and 5 on the French Collection

Analyzing the overall nDCG comparison, we observe that Runs 3 and 5 consistently outperform Run 2 across both short and long-term evaluations. These runs achieve higher nDCG scores, indicating their superior effectiveness in capturing relevant documents. The fact that both Runs 3 and 5 exhibit similar levels of performance suggests comparable retrieval capabilities among these two.

However, when considering the RnD values, we observe some variations among the runs. Run 5 demonstrates a noticeable drop in nDCG compared to the heldout run, as indicated by the RnD delta values. This indicates a potential decrease in retrieval performance when transitioning from the heldout period to the short and long term.

On the other hand, Run 3 displays a relatively smaller drop in nDCG compared to the heldout run, suggesting greater stability and consistency in capturing relevant documents over time.

In contrast, Run 2 exhibits relatively lower nDCG scores, particularly in the long-term evaluation. This can be attributed to its more greedy approach, which might compromise its ability to retrieve relevant documents as the collection evolves over time.

6.1.2. Overall *MAP* Comparison

The boxplot shown in Figure 8 presents the overall MAP comparison of Runs 2, 3, and 5 on the French Collection. The notation for the used colors is the same as for Figure 7. From the boxplot, we can observe the distribution of MAP scores for each run and collection type. The height of each box indicates the *Interquartile Range (IQR)*, representing the range of the middle 50% of the data. The horizontal line within each box corresponds to the median value, while the whiskers above and below the box extend to the highest and lowest values within 1.5 times the IQR.
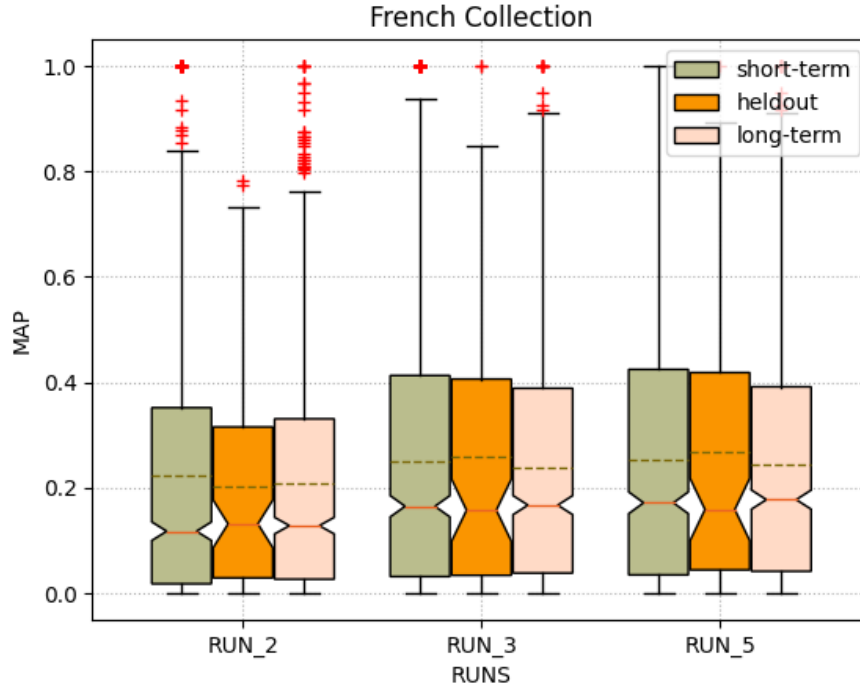
Figure 8: Overall MAP Comparison of Runs 2, 3, and 5 on the French Collection

Analyzing the overall nDCG comparison, we observe that Runs 3 and 5 consistently outperform Run 2 across both short and long-term evaluations, similarly to what we have seen in Section 6.1.1. These runs achieve higher nDCG scores, indicating their superior effectiveness in capturing relevant documents.

However, we can see that Run 5 demonstrates a noticeable drop in nDCG of the long-term compared to the heldout run. This indicates a potential decrease in retrieval performance when transitioning from the heldout period to the long term. On the other hand, Run 3 displays a relatively smaller drop in nDCG compared to the heldout run, suggesting greater stability and consistency in capturing relevant documents over time.

In contrast, Run 2 exhibits relatively lower nDCG scores, particularly in the long-term evaluation. This can be attributed to its more greedy approach, which might compromise its ability to retrieve relevant documents as the collection evolves over time.

Overall, these results confirm what we found in Section 6.1.1: Runs 3 and 5 are competitive with each other, while Run 2 exhibits lower performances compared to these, due to its more basic implementation (see Table 1).

6.1.3. Two-way *ANOVA* on the Long Term French Collection

The results of the two-way ANOVA conducted on the Long Term French Collection are depicted in Figure 9, which showcases the mean nDCG and *Average Precision (AP)* scores for Run 2 in red, Run 3 in grey, and Run 5 in blue. This plot enables us to examine the effects of both the run choice and the long-term evaluation on the performances.
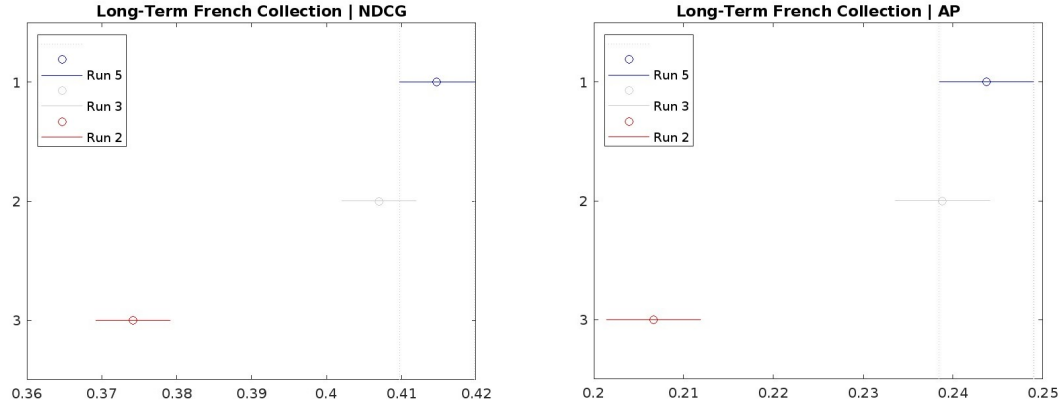
Figure 9: Two-way ANOVA plots for the Runs on the French Collection

Analyzing the two-way ANOVA plot above, at first view we can notice, as expected, that there is no interaction among the three runs.

Then, we observe that Run 5 consistently achieves the highest mean nDCG and AP scores across different evaluation points. Run 2 exhibits lower performance compared to the other runs, while Run 3 shows a relatively stable performance, albeit slightly lower than Run 5, confirming what we found in Section 6.1.1. In addition, the overlap we find among the columns of Run 3 and Run 5 suggests that the small changes done from Run 3 to Run 5 (see Table 1) led to some noticeable albeit small improvements. These findings suggest that both the choice of run and the long-term evaluation have a significant impact on the overall nDCG and AP scores obtained.

To complement the two-way ANOVA plot, we refer to Tables 5 and 6, that present additional statistical information. providing multiple comparisons between the mean nDCG and AP, scores, variance, and p-values associated with runs 2, 3, and 5. The p-values indicate the level of significance and suggest whether there is a significant difference in performance between runs.

The tables present the results of the pairwise comparisons between different runs. The "Group A" and "Group B" columns indicate the runs being compared. The "Lower Limit" and "Upper Limit" columns represent the lower and upper bounds of the confidence interval, while the "A-B" column indicates the mean difference between the runs. The "P-value" column displays the statistical significance of the comparison.

Group A	Group B	Lower Limit	A-B	Upper Limit	P-value
5	3	-0.0023	0.0077	0.0177	0.1661
5	2	0.0305	0.0405	0.0505	1.16E-21
3	2	0.0228	0.0328	0.0428	3.84E-14

Table 5: nDCG Multiple Comparisons for Runs 2, 3, and 5 on the Long Term French Collection

Group A	Group B	Lower Limit	A-B	Upper Limit	P-value
5	3	-0.0057	0.0049	0.0154	0.523
5	2	0.0265	0.037	0.0476	0
3	2	0.0216	0.0322	0.0427	0

Table 6: AP Multiple Comparisons for Runs 2, 3, and 5 on the Long Term French Collection

Looking at Table 5, the comparison between Run 5 and Run 3 yields a p-value of 0.1661, indicating that the mean difference in nDCG scores between these runs is not statistically significant. Similarly, the comparison between Run 5 and Run 2 results in an extremely low p-value of 1.16E-21, suggesting a highly significant difference in nDCG scores. Finally, the comparison between Run 3 and Run 2 also demonstrates a remarkably low p-value of 3.84E-14, indicating a significant discrepancy in their nDCG scores.

These statistical results from the multiple comparisons provide further evidence of the differences in the retrieval performance among the runs on the Long Term French Collection. The significance of the mean differences confirms that the choice of run significantly affects the nDCG scores. Additionally, the p-values obtained from the comparisons reinforce the observations made from the two-way ANOVA plot, highlighting the superior performance of Run 5 and the relatively stable performance of Run 3 compared to the other runs.

6.1.4. Two-way ANOVA on the Short Term French Collection

Similarly, the results of the two-way ANOVA for mean nDCG and AP performed on the Short Term French Collection are visualized in Figure 10.
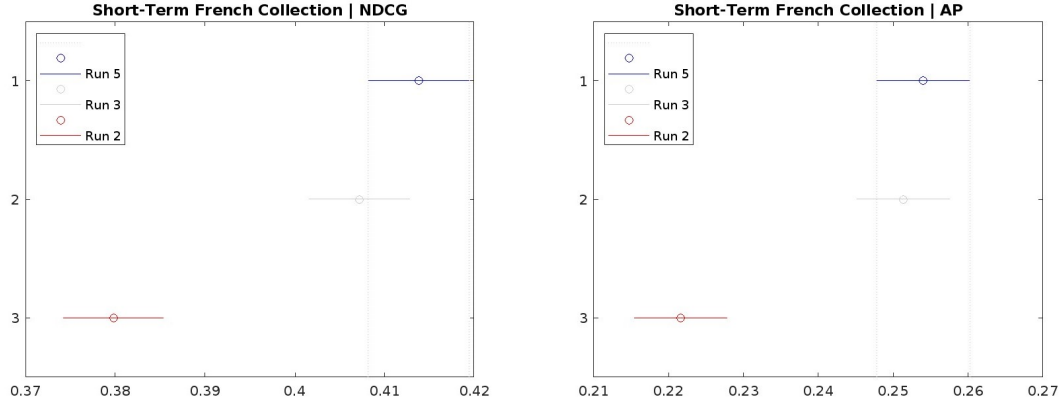
Figure 10: Two-way ANOVA for Runs 2, 3, and 5 on the Short Term French Collection

The considerations made in Section 6.1.3 hold in almost the same way for the results the system got on short-term Collection.

To further investigate the statistical differences among the runs, we refer to the multiple comparisons Tables 7 and 8 shown below, which uses the same notation as Table 5:

Group A	Group B	Lower Limit	A-B	Upper Limit	P-value
5	3	-0.0047	0.0066	0.0178	0.355
5	2	0.0227	0.034	0.0452	3.86E-12
3	2	0.0162	0.0274	0.0386	3.33E-08

Table 7: nDCG Multiple Comparisons for Runs 2, 3, and 5 on the Short Term French Collection

Group A	Group B	Lower Limit	A-B	Upper Limit	P-value
5	3	-0.0098	0.0027	0.0151	0.8701
5	2	0.0199	0.0324	0.0448	3.15×10^{-9}
3	2	0.0173	0.0297	0.0422	6.48×10^{-8}

Table 8: AP Multiple Comparisons for Runs 2, 3, and 5 on the Long Term French Collection

Looking at Table 7, the comparison between Run 5 and Run 3 yields a p-value of 0.355, indicating that the mean difference in nDCG scores between these runs is not statistically significant. Similarly, the comparison between Run 5 and Run 2 results in a remarkably low p-value of 3.86E-12, suggesting a highly significant difference in nDCG scores. Finally, the comparison between Run 3 and Run 2 also demonstrates a low p-value of 3.33E-08, indicating a significant discrepancy in their nDCG scores.

In summary, the statistical analysis of the results obtained on the French collection reveals valuable insights into the performance of different runs. Run 5 has been multiple times confirmed to be the most effective one among the three, showing a noticeable overall strength in retrieving and ranking relevant documents. This was easily predictable since this is the last and most elaborate and promising implementation we did for the French Collection, as discussed in Section 5.

6.2. Analysis of Results on the English Collection

6.2.1. Overall *nDCG* Comparison

The overall nDCG comparison for Runs 1 and 4 on the English collection is depicted in Figure 11, which uses the same notation as Figure 7.
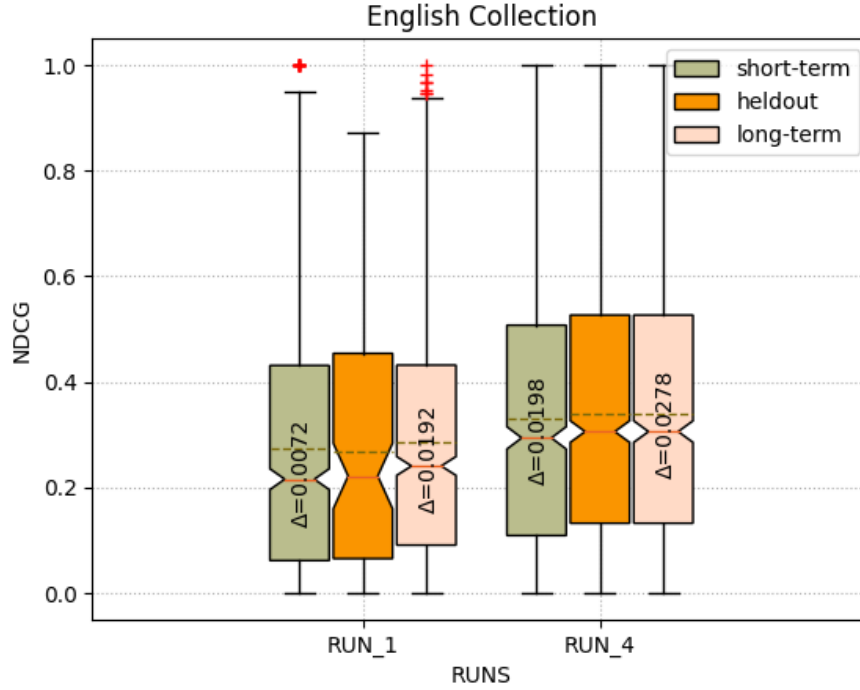
Figure 11: Overall nDCG Comparison for Runs 1 and 4 on the English Collection

Upon analyzing the overall nDCG comparison, we first notice that the results on the English Collection are notably worse compared to those observed for the French Collection in Section 6.1.1. This discrepancy is not unexpected, since the focus of our work was primarily on improving the retrieval performance of our IR system on the French Collection.

Examining the boxplot in Figure 11, we can observe that Run 4 consistently outperforms Run 1 and achieves higher nDCG scores across both the Short and Long Term evaluations on the English Collection. The clear separation between the two runs in the boxplot suggests significant differences in their retrieval performance, indicating the superior effectiveness of Run 4 in retrieving relevant documents.

Furthermore, by considering the RnD deltas represented by the values inside the boxes, we can observe that Run 4 exhibits a good level of stability between the Short and Long Term evaluations, with minimal changes in its nDCG scores. On the other hand, Run 1 shows a notable increase in RnD from the Short to the Long Term, more than doubling the delta value. This indicates that Run 1's performance significantly deteriorates when transitioning from the Short to the Long Term evaluation of the English Collection.

In summary, the overall nDCG comparison confirms that Run 4 performs better than Run 1 in all evaluation settings on the English Collection. Run 4 demonstrates higher nDCG scores and exhibits greater stability between the Short and Long Term evaluations, while Run 1 experiences a noticeable decline in performance when moving from the Short to the Long Term. These findings highlight the importance of the improvements achieved in Run 4 and emphasize its superior retrieval effectiveness compared to Run 1.

6.2.2. Overall *MAP* Comparison

We now turn our attention to the overall MAP comparison of Runs 1 and 4 on the English collection, as illustrated in Figure 12. The boxplot showcases the distribution of MAP scores, using the same notation as Figure 8.
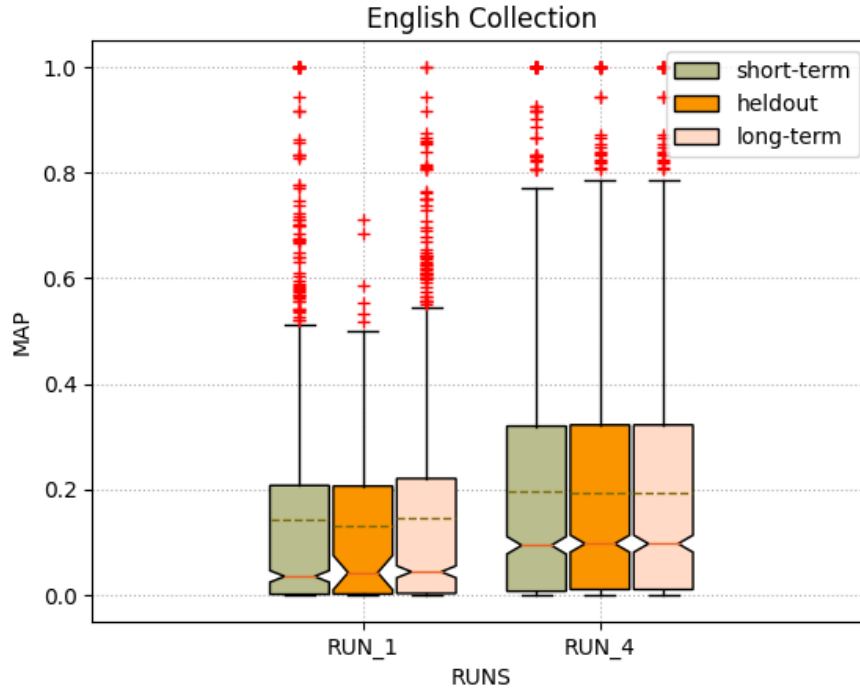
Figure 12: Overall MAP Comparison of Runs 1 and 4 on the English Collection

Analyzing the boxplot in Figure 12, we observe that Run 4 consistently outperforms Run 1 on all three collections: Short Term, Heldout, and Long Term. Run 4 exhibits higher median MAP scores in both the Short Term and Long Term evaluations, indicating its superior retrieval performance in terms of average precision across different evaluation points.

Although the distribution of MAP scores for Run 1 appears to be more concentrated, suggesting more consistent performance, Run 4 clearly demonstrates better overall effectiveness in retrieving relevant documents on the English Collection.

However, in terms of the Heldout Collection, both Runs 1 and 4 display comparable median MAP scores, indicating similar retrieval performance within this collection.

These findings reinforce the conclusions drawn in Section 6.2.1 regarding the superior performance of Run 4 compared to Run 1 on the English Collection.

6.2.3. Two-way *ANOVA* on the Long Term English Collection

The results of the two-way ANOVA performed on the Long Term English Collection are shown in Figure 13, which illustrates the mean nDCG and AP scores for Run 1 in red on the left and Run 4 in blue on the right.
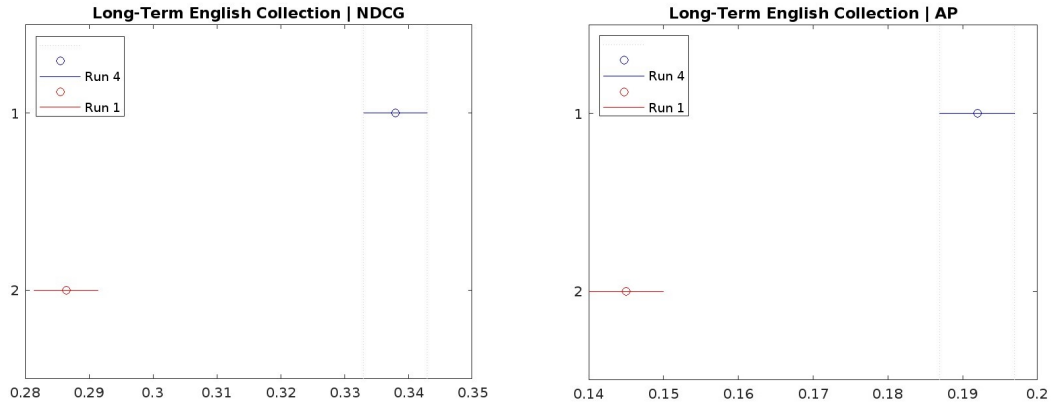
Figure 13: Two-way ANOVA for Runs 1 and 4 on the Long Term English Collection

At first, we can see that, as expected, there is no interaction between these two runs. To reinforce this, we can also see that there is no overlap between their column means.

Continuing into analyzing the two-way ANOVA plot, we find that Run 4 achieves higher mean nDCG scores compared to Run 1 across different evaluation points. This suggests the superiority of Run 4 in retrieving relevant documents in the long term on the English collection, keep confirming what we found in Section 6.2.1.

To provide additional statistical information, we refer to Tables 9 and 10, which present the results of multiple comparisons between Run 1 and Run 4 for the Long Term English Collection. The table shows the lower limit, A-B difference, upper limit, and p-value for the comparison. In this case, there is a single row in the table since there is only one comparison between the two runs.

Group A	Group B	Lower Limit	A-B Difference	Upper Limit	P-value
4	1	0.0415	0.0515	0.0615	6.77×10^{-25}

Table 9: nDCG Multiple comparisons for the Long Term English Collection

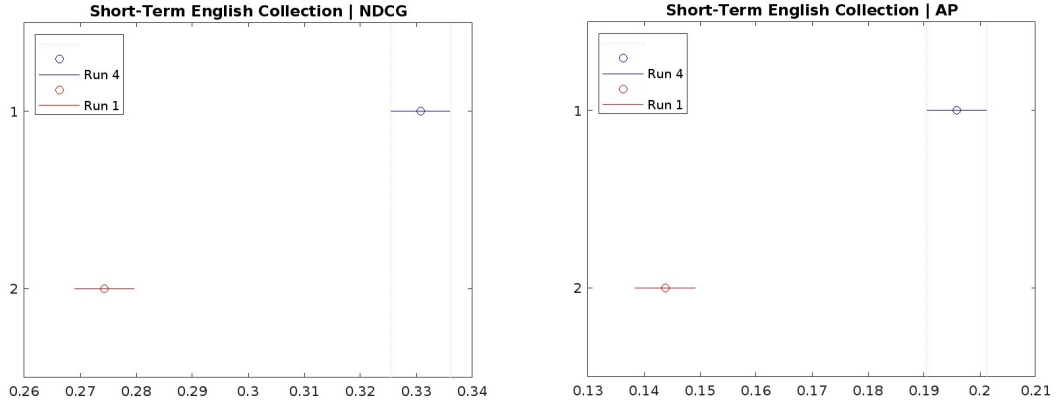| Group A	Group B	Lower Limit	A-B Difference	Upper Limit	P-value
4	1	0.0369	0.0469	0.057	2.13×10^{-20}

Table 10: AP Multiple comparisons for the Long Term English Collection

Table 9 suggests a significant difference between Run 1 and Run 4 in terms of nDCG scores for the Long Term English Collection. The positive A-B difference and the large p-value indicate that Run 4 consistently achieves higher nDCG scores than Run 1 across different evaluation points.

These findings further support the conclusion that Run 4 performs better than Run 1 in terms of retrieving relevant documents over the long term on the English collection, as discussed in Sections 6.2.1 and 6.2.2.

6.2.4. Two-way *ANOVA* on the Short Term English Collection

Similarly, the results of the two-way ANOVA for mean nDCG and AP performed on the Short Term English Collection are depicted in Figure 14.

Figure 14: Two-way ANOVA for Runs 1 and 4 on the Short Term English Collection

These results confirm everything we have found until now, indicating the superiority of run 4 on both Short Term and Long Term Collection.

To provide additional statistical information, Tables 11 and 12 present the results of the multiple comparisons between Run 1 and Run 4.

Group A	Group B	Lower Limit	A-B Difference	Upper Limit	P-value
4	1	0.0457	0.0564	0.0671	0.00×10^{0}

Table 11: Multiple comparisons for the Short Term English Collection

Group A	Group B	Lower Limit	A-B Difference	Upper Limit	P-value
4	1	0.0412	0.052	0.0628	1.05×10^{-21}

Table 12: Multiple comparisons for the Short Term English Collection

Table 11 shows that there is a significant difference in the mean nDCG scores between Run 1 and Run 4 on the Short Term English Collection. The A-B difference, which represents the difference in nDCG scores between the two runs, is 0.0564, indicating that Run 4 performs substantially better than Run 1. The p-value, which is practically zero, further supports the significance of this difference.

These findings consistently demonstrate the superiority of Run 4 over Run 1 in terms of nDCG scores in both short and long-term evaluations of the English collection.

6.3. Final Considerations on Statistical Analysis

In this section, we conducted a comprehensive statistical analysis of the retrieval performance of our system on both the French and English collections. We focused on evaluating the overall nDCG and MAP scores, as well as conducting two-way ANOVA tests to examine the effects of different factors on the performance.

For the French collection, we compared the performances of Runs 2, 3, and 5. The analysis revealed that Run 5 consistently exhibited slightly better or comparable performance compared to Run 3 across all evaluation points. This suggests that the modifications made in Run 5 resulted in improved retrieval effectiveness, making it a promising option. On the other hand, Run 2 displayed significantly worse performance compared to the other runs, indicating the need for further improvements.

Turning to the English collection, we focused on comparing Runs 1 and 4. The results consistently demonstrated that Run 4 outperformed Run 1 in every aspect of the analysis. It achieved higher nDCG scores, indicating its superior effectiveness in retrieving relevant documents. Additionally, Run 4 displayed a higher median MAP score, indicating its superior overall performance in both short and long windows of time. These findings establish Run 4 as the more successful option for retrieval on the English collection.

The two-way ANOVA tests further supported our conclusions. In the French collection, the two-way ANOVA analysis confirmed the superiority of Run 5 over Run 3 in terms of nDCG scores. This finding aligns with the overall comparison results.

In the English collection, the two-way ANOVA did not provide any new unexpected result, as the performance differences between Run 1 and Run 4 were consistently evident in all previous aspects of the analysis.

In summary, our statistical analysis highlights the varying performances of different runs in the French and English collections. For the French collection, Run 5 exhibited slightly better or comparable performance compared to Run 3, while Run 2 displayed significantly worse performance. In contrast, for the English collection, Run 4 consistently outperformed Run 1 in every aspect of the analysis.

7. Conclusions and Future Work

In this work, we presented our approach to the CLEF *Long Eval LAB 2023* task, which aimed to develop an effective and efficient search engine for web documents.

Our approach consisted of using a combination of different techniques, including query expansion, re-ranking, and the use of large language models such as *ChatGPT* and *SBERT*.

Our experiments showed that our approach achieved good results in terms of effectiveness and efficiency, outperforming the baseline system provided by CLEF. Specifically, we found that combining two different scores in the re-ranking phase led to significant improvements in the retrieval performance. Moreover, we identified several areas for future work that could further improve the effectiveness and efficiency of our approach.

One possible direction for future work is to find better ways to combine scores or add other scores to the re-ranking phase. We plan to explore different combinations of scores and investigate the use of other large language models, such as other available *BERT* models trained, or to train some specifically for this task.

Another area for future work is to find better prompts [44] to use in *ChatGPT* for improving query expansion. We also plan to investigate the use of other *Large Language Model (LLM)* techniques for query expansion.

We also want to explore ways to increase the similarity in *SBERT* [41], to increase the number of relevant documents found in the re-ranking phase. One possible approach is to fine-tune the *SBERT* [41] model on our specific task.

Another direction for future work is to index documents as vectors and use them directly, instead of calculating them in re-ranking. This trade-off would result in the loss of one of the scores, but it would increase the re-ranking speed.

Finally, we plan to use links inside documents to extract details that may improve the searching results. We may try to find keywords in the URL path and use them to find their domain authority and take this aspect into account in the score computation.

References

[1] C. Organizers, Longeval clef 2023 lab, https://clef-longeval.github.io/, 2023. Accessed: 2023-05-20.

[2] Qwant, About qwant, https://about.qwant.com/en/, 2023. Accessed: 2023-05-20.

[3] Nicola Ferro, Hellotipster, Bitbucket repository, 2023. URL: https://bitbucket.org/frrncl/se-unipd/src/master/hello-tipster/.

[4] A. Lucene, Apache lucene, https://lucene.apache.org/, 2023. Accessed: 2023-05-20.

[5] M. D. Network, Regular expressions, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions, 2023. Accessed: 2023-05-20.

[6] A. Lucene, Lucene analyzer, https://lucene.apache.org/core/8_0_0/core/org/apache/lucene/analysis/Analyzer.html, 2023. Accessed: 2023-05-20.

[7] A. Lucene, Lucene whitespacetokenizer, https://lucene.apache.org/core/7_4_0/analyzers-common/org/apache/lucene/analysis/core/WhitespaceTokenizer.html, 2023. Accessed: 2023-05-20.

[8] A. Lucene, Lucene lettertokenizer, https://lucene.apache.org/core/7_3_1/analyzers-common/org/apache/lucene/analysis/core/LetterTokenizer.html, 2023. Accessed: 2023-05-20.

[9] A. Lucene, Lucene standardtokenizer, https://lucene.apache.org/core/6_6_0/core/org/apache/lucene/analysis/standard/StandardTokenizer.html, 2023. Accessed: 2023-05-20.

[10] A. S. Foundation, Apache solr, https://solr.apache.org/, 2023. Accessed: 2023-05-20.

[11] A. S. Foundation, Apache solr englishminimalstemfilter, https://solr.apache.org/guide/6_6/filter-descriptions.html#FilterDescriptions-EnglishMinimalStemFilter, 2023. Accessed: 2023-05-20.

[12] A. S. Foundation, Apache solr kstemfilter, https://solr.apache.org/guide/6_6/filter-descriptions.html#FilterDescriptions-KStemFilter, 2023. Accessed: 2023-05-20.

[13] A. S. Foundation, Apache solr porterstemfilter, https://solr.apache.org/guide/6_6/filter-descriptions.html#FilterDescriptions-PorterStemFilter, 2023. Accessed: 2023-05-20.

[14] A. S. Foundation, Apache solr frenchlightstemfilter, https://solr.apache.org/guide/6_6/language-analysis.html#LanguageAnalysis-FrenchLightStemFilter, 2023. Accessed: 2023-05-20.

[15] A. S. Foundation, Apache solr frenchminimalstemfilter, https://solr.apache.org/guide/6_6/language-analysis.html#LanguageAnalysis-FrenchLightStemFilter, 2023. Accessed: 2023-05-20.

[16] A. Lucene, Lucene lovinsstemmer, https://lucene.apache.org/core/6_3_0/analyzers-common/org/tartarus/snowball/ext/LovinsStemmer.html, n.d. Accessed: 2023-05-20.

[17] A. Lucene, Lucene lengthfilter, https://lucene.apache.org/core/7_0_1/analyzers-common/org/apache/lucene/analysis/miscellaneous/LengthFilter.html, 2023. Accessed: 2023-05-20.

[18] A. Lucene, Lucene englishpossessivefilter, 2023. Accessed: 2023-05-20.

[19] A. Lucene, Lucene stopfilter, 2023. Accessed: 2023-05-20.

[20] G. Code, Luke, https://code.google.com/archive/p/luke/, 2023. Accessed: 2023-05-20.

[21] A. Lucene, Lucene ngramtokenfilter, https://lucene.apache.org/core/8_1_1/ analyzers-common/org/apache/lucene/analysis/ngram/NGramTokenFilter.html, 2023. Accessed: 2023-05-20.

[22] A. Lucene, Lucene shinglefilter, https://lucene.apache.org/core/4_3_0/ analyzers-common/org/apache/lucene/analysis/shingle/ShingleFilter.html, 2023. Accessed: 2023-05-20.

[23] A. S. Foundation, Apache solr opennlp part of speech filter, https://solr.apache.org/ guide/7_3/language-analysis.html#opennlp-part-of-speech-filter, 2023. Accessed: 2023-05-20.

[24] A. S. Foundation, Solr opennlp lemmatizer filter, https://solr.apache.org/guide/7_ 3/language-analysis.html#opennlp-lemmatizer-filter, 2023. Accessed: 2023-05-20.

[25] A. Lucene, Lucene elisionfilter, https://lucene.apache.org/core/7_3_1/ analyzers-common/org/apache/lucene/analysis/util/ElisionFilter.html, 2023. Accessed: 2023-05-20.

[26] A. Lucene, Lucene lowercasefilter, https://lucene.apache.org/core/8_0_0/ analyzers-common/org/apache/lucene/analysis/core/LowerCaseFilter.html, 2023. Accessed: 2023-05-20.

[27] A. Lucene, Lucene asciifoldingfilter, https://lucene.apache.org/core/4_9_0/ analyzers-common/org/apache/lucene/analysis/miscellaneous/ASCIIFoldingFilter. html, 2023. Accessed: 2023-05-20.

[28] A. Lucene, Lucene synonymgraphfilter, https://lucene.apache.org/core/6_4_1/ analyzers-common/org/apache/lucene/analysis/synonym/SynonymGraphFilter. html, 2023. Accessed: 2023-05-20.

[29] A. Lucene, Lucene synonymmap, https://lucene.apache.org/core/7_3_0/ analyzers-common/org/apache/lucene/analysis/synonym/SynonymMap.html, 2023. Accessed: 2023-05-20.

[30] A. Lucene, Lucene bm25similarity, https://lucene.apache.org/core/7_0_1/core/org/ apache/lucene/search/similarities/BM25Similarity.html, 2023. Accessed: 2023-05-20.

[31] H. Face, Hugging face 'all-minilm-l6-v2' model, https://huggingface.co/optimum/ all-MiniLM-L6-v2, 2023. Accessed: 2023-05-20.

[32] S. Wang, H. Scells, G. Zuccon, B. Koopman, Can chatgpt write a good boolean query for systematic review literature search?, Journal of Information Retrieval 1 (2023). Accessed: 2023-05-24.

[33] OpenAI, Openai text completion api documentation, https://platform.openai.com/ docs/guides/completion/introduction, n.d.. Accessed: 2023-05-20.

[34] OpenAI, How do text davinci-002 and text davinci-003 differ?, OpenAI Help Center (n.d.). Accessed: 2023-05-20.

[35] A. Lucene, Lucene booleanquery, https://lucene.apache.org/core/8_1_1/core/org/ apache/lucene/search/BooleanQuery.html, 2023. Accessed: 2023-05-20.

[36] A. Lucene, Lucene boostquery, https://lucene.apache.org/core/7_3_1/core/org/ apache/lucene/search/BoostQuery.html, 2023. Accessed: 2023-05-20.

[37] UKPLab, Sentence transformers, https://github.com/UKPLab/sentence-transformers, 2023. Accessed: 2023-05-20.

[38] J.-P. Fauconnier, French word embeddings, 2015. URL: http://fauconnier.github.io.

[39] Apache Lucene, Lucene knnvectorfield, Online documentation, 2021. URL: https://lucene.apache.org/core/9_0_0/core/org/apache/lucene/document/KnnVectorField.html.

[40] A. Lucene, Lucene multisimilarity, https://lucene.apache.org/core/8_0_0/core/org/apache/lucene/search/similarities/MultiSimilarity.html, n.d. Accessed: 2023-05-20.

[41] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, 2019. URL: https://arxiv.org/abs/1908.10084.

[42] PyTorch, Pytorch cosinesimilarity, Online documentation, 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.CosineSimilarity.html.

[43] National Institute of Standards and Technology, Manhattan distance, Online dictionary, 2019. URL: https://xlinux.nist.gov/dads/HTML/manhattanDistance.html.

[44] S. Wang, H. Scells, B. Koopman, G. Zuccon, Can chatgpt write a good boolean query for systematic review literature search?, 2023. arXiv:2302.03495.